

How to Meet Asynchronously at Polynomial Cost

Yoann Dieudonné*

Andrzej Pelc†

Vincent Villain‡

January 31, 2013

Abstract

Two mobile agents starting at different nodes of an unknown network have to meet. This task is known in the literature as *rendezvous*. Each agent has a different label which is a positive integer known to it, but unknown to the other agent. Agents move in an asynchronous way: the speed of agents may vary and is controlled by an adversary. The cost of a rendezvous algorithm is the total number of edge traversals by both agents until their meeting. The only previous deterministic algorithm solving this problem has cost exponential in the size of the graph and in the larger label. In this paper we present a deterministic rendezvous algorithm with cost *polynomial* in the size of the graph and in the *length* of the *smaller* label. Hence we decrease the cost exponentially in the size of the graph and doubly exponentially in the labels of agents.

As an application of our rendezvous algorithm we solve several fundamental problems involving teams of unknown size larger than 1 of labeled agents moving asynchronously in unknown networks. Among them are the following problems: **team size**, in which every agent has to find the total number of agents, **leader election**, in which all agents have to output the label of a single agent, **perfect renaming** in which all agents have to adopt new different labels from the set $\{1, \dots, k\}$, where k is the number of agents, and **gossiping**, in which each agent has initially a piece of information (value) and all agents have to output all the values. Using our rendezvous algorithm we solve all these problems at cost polynomial in the size of the graph and in the smallest length of all labels of participating agents.

keywords: asynchronous mobile agents, network, rendezvous, deterministic algorithm, leader election, renaming, gossiping

*MIS, Université de Picardie Jules Verne Amiens, France. E-mail: yoann.dieudonne@u-picardie.fr

†Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada. pelc@uqo.ca. Partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

‡MIS, Université de Picardie Jules Verne Amiens, France. E-mail: vincent.villain@u-picardie.fr

1 Introduction

The background. Two mobile agents, starting at different nodes of a network, possibly at different times, have to meet. This basic task, known as *rendezvous*, has been thoroughly studied in the literature. It even has applications in human and animal interaction, e.g., when agents are people that have to meet in a city whose streets form a network, or migratory birds have to gather at one destination flying in from different places. In computer science applications, mobile agents usually represent software agents in computer networks, or mobile robots, if the network is a labyrinth or is composed of corridors in a building. The reason to meet may be to exchange data previously collected by the agents, or to coordinate some future task, such as network maintenance or finding a map of the network.

In this paper we consider the rendezvous problem under a very weak scenario which assumes little knowledge and control power of the agents. This makes our solutions more widely applicable, but significantly increases the difficulty of meeting. More specifically, agents do not have any a priori information about the network, they do not know its topology or any bounds on parameters such as the diameter or the size. We seek rendezvous algorithms that do not rely on the knowledge of node labels, and can work in anonymous networks as well (cf. [5]). The importance of designing such algorithms is motivated by the fact that, even when nodes are equipped with distinct labels, agents may be unable to perceive them because of limited sensory capabilities, or nodes may refuse to reveal their labels, e.g., due to security or privacy reasons. Note that if nodes had distinct labels that can be perceived by the agents, then agents might explore the network and meet in the smallest node, hence rendezvous would reduce to exploration. Agents have distinct labels, which are positive integers and each agent knows its own label, but not the label of the other agent. The label of the agent is the only a priori initial input to its algorithm. During navigation agents gain knowledge of the visited part of the network: when an agent enters a node, it learns the port number by which it enters and the degree of the node. The main difficulty of the scenario is the asynchronous way in which agents move: the speed of the agents may vary, may be different for each of them, and is totally controlled by an adversary. This feature of the model is also what makes it more realistic than the synchronous scenario: in practical applications the speed of agents depends on various factors that are beyond their control, such as congestion in different parts of the network or mechanical characteristics in the case of mobile robots. Notice that in the asynchronous scenario we cannot require that agents meet in a node: the adversary can prevent this even in the two-node graph. Thus, similarly as in previous papers on asynchronous rendezvous [10, 16, 18, 19, 28], we allow the meeting either in a node or inside an edge. The cost of a rendezvous algorithm is the total number of edge traversals by both agents until their meeting.

Our results. The main result of this paper is a deterministic rendezvous algorithm, working in arbitrary unknown networks and whose cost is polynomial in the size of the network and in the *length* of the *smaller* label (i.e. in the logarithm of this label). The only previous algorithm solving the asynchronous rendezvous problem [18] is exponential in the size of the network and in the larger label. Hence we decrease the cost exponentially in the size of the network and doubly exponentially in the labels of agents.

As an application of our rendezvous algorithm we solve several fundamental problems involving teams of unknown size larger than 1 of labeled agents moving asynchronously in unknown networks. Among them are the following problems: **team size**, in which every agent has to find the total number of agents, **leader election**, in which all agents have to output the label of a single agent, **perfect renaming** in which all agents have to adopt new different labels from the set $\{1, \dots, k\}$,

where k is the number of agents, and **gossiping**, in which each agent has initially a piece of information (value) and all agents have to output all the values. Using our rendezvous algorithm we solve all these problems at cost (total number of edge traversals by all agents) polynomial in the size of the graph and in the smallest length of all labels of participating agents. To the best of our knowledge this is the first solution of these problems for asynchronous mobile agents, even regardless of the cost.

The model. The network is modeled as a finite undirected connected graph, referred to hereafter as a graph. Nodes are unlabeled, but edges incident to a node v have distinct labels in $\{0, \dots, d-1\}$, where d is the degree of v . Thus every undirected edge $\{u, v\}$ has two labels, which are called its *port numbers* at u and at v . Port numbering is *local*, i.e., there is no relation between port numbers at u and at v . Note that in the absence of port numbers, edges incident to a node would be undistinguishable for agents and thus gathering would be often impossible, as the adversary could prevent an agent from taking some edge incident to the current node.

In order to avoid crossings of non-incident edges, we consider an embedding of the underlying graph in the three-dimensional Euclidean space, with nodes of the graph being points of the space and edges being pairwise disjoint line segments joining them. Agents are modeled as points moving inside this embedding. (This embedding is only for the clarity of presentation; in fact crossings of non-incident edges would make rendezvous simpler, as agents traversing distinct edges could sometimes meet accidentally at the crossing point.)

There are two agents that start from different nodes of the graph and traverse its edges. They cannot mark visited nodes or traversed edges in any way. Agents have distinct labels which are strictly positive integers. Each agent knows only its own label which is an initial input to its deterministic algorithm. Agents do not know the topology of the graph or any bound on its size. They can, however, acquire knowledge about the network: When an agent enters a node, it learns its degree and the port of entry. We assume that the memory of the agents is unbounded: from the computational point of view they are modeled as Turing machines.

Agents navigate in the graph in an asynchronous way which is formalized by an adversarial model used in [10, 16, 18, 19, 28] and described below. Two important notions used to specify movements of agents are the *route* of the agent and its *walk*. Intuitively, the agent chooses the route *where* it moves and the adversary describes the walk on this route, deciding *how* the agent moves. More precisely, these notions are defined as follows. The adversary initially places an agent at some node of the graph. The route is chosen by the agent and is defined as follows. The agent chooses one of the available ports at the current node. After getting to the other end of the corresponding edge, the agent chooses one of the available ports at this node or decides to stay at this node. It does so on the basis of all information currently available to it. The resulting route of the agent is the corresponding sequence of edges $(\{v_0, v_1\}, \{v_1, v_2\}, \dots)$, which is a (not necessarily simple) path in the graph.

We now describe the walk f of an agent on its route. Let $R = (e_1, e_2, \dots)$ be the route of an agent. Let $e_i = \{v_{i-1}, v_i\}$. Let (t_0, t_1, t_2, \dots) , where $t_0 = 0$, be an increasing sequence of reals, chosen by the adversary, that represent points in time. Let $f_i : [t_i, t_{i+1}] \rightarrow [v_i, v_{i+1}]$ be any continuous function, chosen by the adversary, such that $f_i(t_i) = v_i$ and $f_i(t_{i+1}) = v_{i+1}$. For any $t \in [t_i, t_{i+1}]$, we define $f(t) = f_i(t)$. The interpretation of the walk f is as follows: at time t the agent is at the point $f(t)$ of its route. This general definition of the walk and the fact that (as opposed to the route) it is designed by the adversary, are a way to formalize the asynchronous characteristics of the process. The movement of the agent can be at arbitrary speed, the adversary

may sometimes stop the agent or move it back and forth, as long as the walk in each edge of the route is continuous and covers all of it. This definition makes the adversary very powerful, and consequently agents have little control on how they move. This makes a meeting between agents hard to achieve. Agents with routes R_1 and R_2 and with walks f_1 and f_2 meet at time t , if points $f_1(t)$ and $f_2(t)$ are identical. A meeting is guaranteed for routes R_1 and R_2 , if the agents using these routes meet at some time t , regardless of the walks chosen by the adversary.

Related work. In most papers on rendezvous a synchronous scenario was assumed, in which agents navigate in the graph in synchronous rounds. An extensive survey of randomized rendezvous in various scenarios can be found in [5], cf. also [3, 4, 6, 7]. Deterministic rendezvous in networks has been surveyed in [33]. Several authors considered the geometric scenario (rendezvous in an interval of the real line, see, e.g., [11], or in the plane, see, e.g., [8, 9]). Rendezvous of more than two agents, often called gathering, has been studied, e.g., in [21, 22, 32, 37]. In [21] agents were anonymous, while in [37] the authors considered gathering many agents with unique labels. Gathering many labeled agents in the presence of Byzantine agents was studied in [22]. The problem was also studied in the context of multiple robot systems, cf. [14, 23], and fault tolerant gathering of robots in the plane was studied, e.g., in [2, 15].

For the deterministic setting a lot of effort has been dedicated to the study of the feasibility of rendezvous, and to the time required to achieve this task, when feasible. For instance, deterministic rendezvous with agents equipped with tokens used to mark nodes was considered, e.g., in [31]. Deterministic rendezvous of two agents that cannot mark nodes but have unique labels was discussed in [20, 29, 36]. These papers are concerned with the time of synchronous rendezvous in arbitrary graphs. In [20] the authors show a rendezvous algorithm polynomial in the size of the graph, in the length of the shorter label and in the delay between the starting time of the agents. In [29, 36] rendezvous time is polynomial in the first two of these parameters and independent of the delay.

Memory required by two anonymous agents to achieve deterministic rendezvous has been studied in [25, 26] for trees and in [17] for general graphs. Memory needed for randomized rendezvous in the ring is discussed, e.g., in [30].

Asynchronous rendezvous of two agents in a network has been studied in [10, 16, 18, 19, 28]. The model used in the present paper has been introduced in [19]. In this paper the authors investigated the cost of rendezvous in the infinite line and in the ring. They also proposed a rendezvous algorithm for an arbitrary graph with a known upper bound on the size of the graph. This assumption was subsequently removed in [18], but both in [19] and in [18] the cost of rendezvous was exponential in the size of the graph and in the larger label. In [28] asynchronous rendezvous was studied for anonymous agents and the cost was again exponential. The only asynchronous rendezvous algorithms at polynomial cost were presented in [10, 16], but in these papers authors restricted attention to infinite multidimensional grids and they used the powerful assumption that each agent knows its starting coordinates. (The cost in this case is polynomial in the initial distance).

A different asynchronous scenario was studied in [13, 23] for the plane. In these papers the authors assumed that agents are memoryless, but they can observe the environment and make navigation decisions based on these observations.

The four problems that we solve in the context of asynchronous mobile agents as an application of our rendezvous algorithm, are widely researched tasks in distributed computing, under many scenarios. Counting the number of agents is a basic task, cf. [27], as many mobile agents algorithms depend on this knowledge. Leader election, cf. [34], is a fundamental problem in distributed computing. Renaming was introduced in [1] and further studied by many authors. Gossiping, also

called all-to-all communication, is one of the basic primitives in network algorithms, cf. [24].

2 Preliminaries

Throughout the paper, the number of nodes of a graph is called its size. In this section we recall two procedures known from the literature, that will be used as building blocks in our algorithms. The aim of both of them is graph exploration, i.e., visiting all nodes and traversing all edges of the graph by a single agent. The first procedure, based on universal exploration sequences (UXS), is a corollary of the result of Reingold [35]. Given any positive integer n , it allows the agent to traverse all edges of any graph of size at most n , starting from any node of this graph, using $P(n)$ edge traversals, where P is some polynomial. (The original procedure of Reingold only visits all nodes, but it can be transformed to traverse all edges by visiting all neighbors of each visited node before going to the next node.) After entering a node of degree d by some port p , the agent can compute the port q by which it has to exit; more precisely $q = (p + x_i) \bmod d$, where x_i is the corresponding term of the UXS.

A *trajectory* is a sequence of nodes of a graph, in which each node is adjacent to the preceding one. (Hence it is a sequence of nodes visited following a route.) Given any starting node v , we denote by $R(n, v)$ the trajectory obtained by Reingold's procedure. The procedure can be applied in any graph starting at any node, giving some trajectory. We say that the agent *follows* a trajectory if it executes the above procedure used to construct it. This trajectory will be called *integral*, if the corresponding route covers all edges of the graph. By definition, the trajectory $R(n, v)$ is integral if it is obtained by Reingold's procedure applied in any graph of size at most n starting at any node v .

The second procedure [12] is performed by an agent using a fixed token placed at the starting node of the agent. (It is well known that a terminating exploration even of all anonymous rings of unknown size by a single agent without a token is impossible.) In our applications the roles of the token and of the exploring agent will be played by agents. The procedure works at cost polynomial in the size of the graph. Moreover, at the end of it the agent is with the token and has a complete map of the graph with all port numbers marked. We call this procedure *EST*, for *exploration with a stationary token*. We denote by $T(EST(n))$ the maximum number of edge traversals in an execution of the procedure *EST* in a graph of size at most n .

For a positive integer x , by $|x|$ we denote the length of its binary representation, called the length of x . Hence $|x| = \lceil \log x \rceil$. All logarithms are with base 2. For two agents, we say that the agent with larger (smaller) label is larger (resp. smaller). For any trajectory $T = (v_0, \dots, v_r)$, we denote by \bar{T} the reverse trajectory (v_r, \dots, v_0) . For two trajectories $T_1 = (v_0, \dots, v_r)$ and $T_2 = (v_r, v_{r+1}, \dots, v_s)$ we denote by $T_1 T_2$ the trajectory $(v_0, \dots, v_r, v_{r+1}, \dots, v_s)$. For any trajectory $T = (v_0, \dots, v_r)$, for which $v_r = v_0$ and for any positive integer x , we define T^x to be $TT \dots T$, with x copies of T . For any trajectory T we define $|T|$ to be the number of nodes in T .

3 The rendezvous algorithm

In this section we describe and analyze our rendezvous algorithm working at polynomial cost. Its high-level idea is based on the following observation. If one agent follows an integral trajectory during some time interval, then it must either meet the other agent or this other agent must perform at least one complete edge traversal during this time interval, i.e., it must make *progress*. A naive

use of this observation leads to the following simple algorithm: an agent with label L starting at node v of a graph of size n follows the trajectory $R(n, v)^{(P(n)+1)^L}$ and stops. Indeed, in this case the number of integral trajectories $R(n, v)$ performed by the larger agent is larger than the number of edges traversed by the smaller agent and consequently, if they have not met before, the larger agent must meet the smaller one after the smaller agent stops, because the larger agent will still perform at least one entire trajectory afterwards. However, this simple algorithm has two major drawbacks. First, it requires knowledge of n (or of an upper bound on it) and second, it is exponential in L , while we want an algorithm *polylogarithmic* in L . Hence the above observation has to be used in a much more subtle way. Our algorithm constructs a trajectory for each agent, polynomial in the size of the graph and polylogarithmic in the shorter label, i.e., polynomial in its length, which has the following *synchronization* property that holds in a graph of arbitrary unknown size. When one of the agents has already followed some part of its trajectory, it has either met the other agent, or this other agent must have completed some other related part of its trajectory. (In a way, if the meeting has not yet occurred, the other agent has been “pushed” to execute some part of its route.) The trajectories are designed in such a way that, unless a meeting has already occurred, the agents are forced to follow in the same time interval such parts of their trajectories that meeting is inevitable. A design satisfying this synchronization property is difficult due to the arbitrary behavior of the adversary and is the main technical challenge of the paper.

3.1 Formulation of the algorithm

We first define several trajectories based on trajectories $R(k, v)$. Each trajectory is defined using a starting node v and a parameter k . Notice that, similarly as the basic trajectory $R(k, v)$, each of these trajectories (of increasing complexity) can be defined in any graph, starting from any node v .

Definition 3.1 *The trajectory $X(k, v)$ is the sequence of nodes $R(k, v)\overline{R(k, v)}$.*

Definition 3.2 *The trajectory $Q(k, v)$ is the sequence of nodes $X(1, v)X(2, v) \dots X(k, v)$.*

Definition 3.3 *Let $R(k, v_1) = (v_1, v_2, \dots, v_s)$. Let*

$$Y'(k, v_1) = Q(k, v_1)(v_1, v_2)Q(k, v_2)(v_2, v_3)Q(k, v_3) \dots (v_{s-1}, v_s)Q(k, v_s).$$

We define the trajectory $Y(k, v_1)$ as $Y'(k, v_1)\overline{Y'(k, v_1)}$.

Definition 3.4 *The trajectory $Z(k, v)$ is the sequence of nodes $Y(1, v)Y(2, v) \dots Y(k, v)$.*

Definition 3.5 *Let $R(k, v_1) = (v_1, v_2, \dots, v_s)$. Let*

$$A'(k, v_1) = Z(k, v_1)(v_1, v_2)Z(k, v_2)(v_2, v_3)Z(k, v_3) \dots (v_{s-1}, v_s)Z(k, v_s).$$

We define the trajectory $A(k, v_1)$ as $A'(k, v_1)\overline{A'(k, v_1)}$.

Definition 3.6 *The trajectory $B(k, v)$ is the sequence of nodes $Y(k, v)^{2|A(4k)|}$.*

Definition 3.7 *The trajectory $K(k, v)$ is the sequence of nodes $X(k, v)^{2(|B(4k)|+|A(8k)|)}$.*

Definition 3.8 *The trajectory $\Omega(k, v)$ is the sequence of nodes $X(k, v)^{(2k-1)|K(k)|}$.*

If the node v is clear from the context, we will sometimes omit it, thus writing $X(k)$ instead of $X(k, v)$, etc.

Using the above defined trajectories we describe Algorithm RV-asynch-poly executed by an agent with label L in an arbitrary graph. The agent first modifies its label. If $x = (c_1 \dots c_r)$ is the binary representation of L , define the *modified label* of the agent to be the sequence $M(x) = (c_1 c_1 c_2 c_2 \dots c_r c_r 01)$. Note that, for any x and y , the sequence $M(x)$ is never a prefix of $M(y)$. Also, $M(x) \neq M(y)$ for $x \neq y$.

Algorithm RV-asynch-poly.

Let x be the binary representation of the label L of the agent and let $M(x) = (b_1 b_2 \dots b_s)$. Let v be the starting node of the agent.

Execute until rendezvous.

$i = 1$;

$k = 1$;

repeat

while $i \leq \min(k, s)$ **do**

if $b_i = 1$ **then** follow the trajectory $B(2k, v)^2$

else follow the trajectory $A(4k, v)^2$

if $\min(k, s) > i$ **then** follow the trajectory $K(k, v)$

else follow the trajectory $\Omega(k, v)$

$i := i + 1$

$i := 1$

$k := k + 1$

3.2 Proof of correctness and cost analysis

We will use the following terminology referring to parts of the trajectory constructed by Algorithm RV-asynch-poly. The part before the start of $\Omega(1, v)$ is called the *first piece* and is denoted $\mathcal{T}(1)$, the part between the end of $\Omega(1, v)$ and the beginning of $\Omega(2, v)$ is called the *second piece* and is denoted $\mathcal{T}(2)$, etc. In general, the part between the end of $\Omega(i-1, v)$ and the beginning of $\Omega(i, v)$ is called the *i th piece* and is denoted $\mathcal{T}(i)$. The trajectory $\Omega(r, v)$ between pieces $\mathcal{T}(r)$ and $\mathcal{T}(r+1)$, is called the *r th fence*.

Inside each piece, the trajectory $B(2k, v)^2$ and the trajectory $A(4k, v)^2$ are called *segments*. Each of the two trajectories $B(2k, v)$ in the segment $B(2k, v)^2$ and each of the two trajectories $A(4k, v)$ in the segment $A(4k, v)^2$ are called *atoms*. We denote by $S_i(k)$ the segment in the k th piece corresponding to the bit b_i in $M(x)$. Each trajectory $K(k, v)$ is called a *border*. We denote by $K_{j,j+1}(k)$ the border between the segment $S_j(k)$ and the segment $S_{j+1}(k)$.

We start with the following fact that will be often used in the sequel.

Lemma 3.1 *Suppose that agents a and b operate in a graph G . Let v be a node of G and let m be a positive integer. If in some time interval I agent b keeps repeating the trajectory $X(m, v)$ and agent a follows at least one entire trajectory $X(m, v)$, then the agents must meet during time interval I . The lemma remains true when X is replaced by Y .*

Proof: Let $R = R(m, v)$. By definition, $X(m, v) = R\bar{R}$. During the time interval I agent a follows the entire trajectory R at least once. If at the time when a starts following R , agent b is following \bar{R} , then they have to meet before a finishes R because b is on a reverse path with respect to a . If at the time when a starts following R , agent b is also following R , then they are two cases to consider.

Case 1. a completes trajectory R before b or simultaneously.

In this case they must meet because a “catches” b .

Case 2. b completes trajectory R before a .

In this case agent b starts following trajectory \bar{R} before the time when a completes R . Hence agents must meet by the time a completes trajectory R because b is on a reverse path with respect to a .

For Y instead of X the argument is similar. \square

The following five lemmas establish various synchronization properties concerning the execution of the algorithm by the agents. They show that, unless agents have already met before, if one agent executes some part of Algorithm RV-asynch-poly, then the other agent must execute some other related part of it. These lemmas show the interplay of pieces, fences, segments, atoms and borders that are followed by each of the agents: these trajectories are the milestones of synchronization. In all lemmas we suppose that agents a and b execute Algorithm RV-asynch-poly in a graph of size n , and we let l to be the length of the smaller of their modified labels.

Lemma 3.2 *If the agents have not met before, then by the time one of the agents completes its $(n + l + i)$ th fence, then the other agent must have completed its $(i + 1)$ th piece.*

Proof: Suppose that the stated property is false. Without loss of generality assume that agent b is the first to complete its $(n + l + i)$ th fence $\Omega_b(n + l + i)$. When b completed its $(n + l)$ th fence $\Omega_b(n + l)$, agent a must have completed its first piece $\mathcal{T}_a(1)$, otherwise a and b must have met because the trajectory $\Omega_b(n + l)$ contains more integral trajectories $X(n + l)$ than there are nodes in the trajectory $\mathcal{T}_a(1)$. Indeed, according to Algorithm RV-asynch-poly, the number of nodes in $\mathcal{T}_a(1)$ is bounded by $2(|A(4)| + |B(2)|)$, while according to Definitions 3.8 and 3.7, the number of integral trajectories $X(n + l)$ within $\Omega_b(n + l)$ is equal to $(2(n + l) - 1)|k(n + l)| = (2(n + l) - 1)(|B(4(n + l))| + |A(8(n + l))|)|X(n + l)|$, which is larger than $2(|A(4)| + |B(2)|)$ since $n + l \geq 2$.

When b completes its $(n + l + 1)$ th piece $\mathcal{T}_b(n + l + 1)$, agent a must have completed its first fence $\Omega_a(1)$. Suppose not. This implies that while agent b follows $\mathcal{T}_b(n + l + 1)$, agent a must follow only its first fence $\Omega_a(1)$ or a part of it. This fence consists of repeating the trajectory $X(1)$. Agent b follows at some point the trajectory $A(4(n + l + 1))$ or the trajectory $B(2(n + l + 1))$ in its $(n + l + 1)$ th piece $\mathcal{T}_b(n + l + 1)$. By Definitions 3.5 and 3.6, agent b must have completed $X(1, u)$, for any node u of the graph, and hence must have met a , in view of Lemma 3.1 which is a contradiction.

Similarly we prove that when b completes its $(n + l + 1)$ th fence $\Omega_b(n + l + 1)$, agent a must have completed its second piece $\mathcal{T}_a(2)$, and when b completes its $(n + l + 2)$ th piece $\mathcal{T}_b(n + l + 2)$, agent a must have completed its second fence $\Omega_a(2)$. In general, it follows by induction on i that when b completes its $(n + l + i)$ th fence $\Omega_b(n + l + i)$, agent a must have completed its $(i + 1)$ th piece $\mathcal{T}_a(i + 1)$. \square

Lemma 3.3 *Let b be the first agent to complete its $(2(n + l))$ th fence. If the agents have not met before, then during the time segment in which agent b follows its $(2(n + l))$ th fence, agent a follows*

a trajectory included in $r\Omega_a(j)s$, for some fixed j satisfying $n+l+1 \leq j \leq 2(n+l)$, where r is the last atom of its j th piece $\mathcal{T}_a(j)$, $\Omega_a(j)$ is its j th fence, and s is the first atom of its $(j+1)$ th piece $\mathcal{T}_a(j+1)$. This j will be called the index of agent a .

Proof: Consider the time interval I during which agent b follows its $(2(n+l))$ th fence $\Omega_b(2(n+l))$. If during this time interval agent a has not started any fence, it would have to follow a trajectory included in a piece $\mathcal{T}_a(k)$ for some $k \leq 2(n+l)$, because b was the first agent to complete its $(2(n+l))$ th fence. By Definition 3.8, the trajectory $\Omega_b(2(n+l))$ contains more copies of the integral trajectory $R(2(n+l))$ than there are edge traversals done by agent a . Indeed, according to Algorithm RV-asynch-poly, the number of nodes in $\mathcal{T}_a(k)$ is bounded by $(k-1)|K(k)| + k(2(|A(4k)| + |B(2k)|)) < (2k-1)|K(k)|$, which is at most $2(2(n+l)-1)|K(2(n+l))|$ for $k \leq 2(n+l)$, while the number of integral trajectories $X(2(n+l))$ in $\Omega_b(2(n+l))$ is equal to $2(2(n+l)-1)|K(2(n+l))|$. Hence the agents would have met, which is a contradiction.

Hence agent a must have started some fence during the time interval I . By Lemma 3.2, during the time interval I agent a must have started its j th fence $\Omega_a(j)$, for some $j \in \{n+l+1, \dots, 2(n+l)\}$. Moreover, during the time interval I agent a could not have followed the entire last atom r of its j th piece $\mathcal{T}_a(j)$. Indeed, this would mean that during the time interval I agent a has entirely followed either the trajectory $B(2j)$ or the trajectory $A(4j)$. In the first case, since $j \geq n+l+1$, this would imply that during the time interval I , agent a followed an entire trajectory $B(k, v)$, where v is the starting node of a , for $k \geq 2(n+l+1)$, while b followed only all or a part of the trajectory $\Omega_b(2(n+l))$ consisting of repetitions of $X(2(n+l))$. This would force a meeting because, by Definition 3.6, the trajectory $B(k, v)$, for $k \geq 2(n+l+1)$ contains at least one trajectory $X(2(n+l), u)$ for every node u of the graph. In the second case, in view of $j \geq n+l+1$, a meeting would be forced in a similar way, because the trajectory $A(4j, v)$, also contains at least one trajectory $X(2(n+l), u)$ for every node u of the graph.

This shows that agent a has started the last atom r of its j th piece $\mathcal{T}_a(j)$ during the time interval I . Using a similar argument we prove that agent a could not complete the first atom s of its $(j+1)$ th piece during the time interval I . This completes the proof. \square

Lemma 3.4 *Let b be the first agent to complete its $(2(n+l))$ th fence. If the agents have not met before, then by the time agent b completes its $(2(n+l))$ th fence, agent a must have completed the last atom r of its j th piece, where j is the index of agent a .*

Proof: Suppose not. Then, in view of Lemma 3.3, during the time interval when agent b follows its $(2(n+l))$ th fence, the trajectory of a is included in r . However, according to Definition 3.8, the number of integral trajectories $X(2(n+l))$ in $\Omega_b(2(n+l))$ is $2(|A(8(2(n+l)))| + |B(4(2(n+l)))|)$. Moreover, according to Algorithm RV-asynch-poly, the number of nodes in r is less than $|B(2j)| + |A(4j)|$. So, since $j \leq 2(n+l)$ in view of Lemma 3.3, the number of integral trajectories in $\Omega_b(2(n+l))$ is larger than the number of nodes in r . This would force a meeting. \square

Lemma 3.5 *Let b be the first agent to complete its $(2(n+l))$ th fence. If the agents have not met before, then by the time agent b completes the first atom of its segment $S_1(2(n+l)+1)$, agent a must have completed its j th fence $\Omega_a(j)$, where j is the index of agent a .*

Proof: Suppose not. Then, in view of Lemma 3.4, during the time interval when agent b follows the first atom of its segment $S_1(2(n+l)+1)$, the trajectory of a is included in $\Omega_a(j)$. Since the trajectory $\Omega_a(j)$ consists of repetitions of the trajectory $X(j)$ starting at the same node v , and while following the first atom of $S_1(2(n+l)+1)$ agent b followed at least one trajectory $X(j, u)$ for any node u of the graph (because $j < 2(n+l)+1$ by Lemma 3.3), this would force a meeting in view of Lemma 3.1. \square

Lemma 3.6 *Let b be the first agent to complete its $(2(n+l))$ th fence. Let t be the first time at which an agent finishes its $(2(n+l)+1)$ th piece. If the agents do not meet by time t , then the following properties hold, for j denoting the index of agent a .*

- **Property 1.** *Let t' be the time when agent a completes a segment $S_i(j+1)$, if this segment exists. Let t'' be the time when agent b completes the border $K_{i,i+1}(2(n+l)+1)$, if this border exists. Then $t' < t''$.*
- **Property 2.** *Let t' be the time when agent b completes a segment $S_i(2(n+l)+1)$, if this segment exists. Let t'' be the time when agent a completes the border $K_{i,i+1}(j+1)$, if this border exists. Then $t' < t''$.*
- **Property 3.** *Let t' be the time when agent a completes a border $K_{i,i+1}(j+1)$, if this border exists. Let t'' be the time when agent b completes the first atom of the segment $S_{i+1}(2(n+l)+1)$, if this segment exists. Then $t' < t''$.*
- **Property 4.** *Let t' be the time when agent b completes a border $K_{i,i+1}(2(n+l)+1)$, if this border exists. Let t'' be the time when agent a completes the first atom of the segment $S_{i+1}(j+1)$, if this segment exists. Then $t' < t''$.*

Proof: Assume that the agents do not meet by time t . Suppose, for contradiction, that at least one of the above 4 properties is not satisfied and let μ be the smallest value of the index i for which one of these properties is not satisfied. We consider 4 cases.

Case 1. Property 1 is false for $i = \mu$. This implies that b completed its border $K_{\mu,\mu+1}(2(n+l)+1)$ before a completed $S_\mu(j+1)$. Hence agent b has completed $K_{\mu,\mu+1}(2(n+l)+1)$ while agent a was following $S_\mu(j+1)$. Indeed, if agent b completed $K_{\mu,\mu+1}(2(n+l)+1)$ before agent a started $S_\mu(j+1)$, this would imply:

- if $\mu > 1$ then agent b started $K_{\mu,\mu+1}(2(n+l)+1)$ before agent a has completed $K_{\mu-1,\mu}(j+1)$. Hence b had completed $S_\mu(2(n+l)+1)$ before a completed $K_{\mu-1,\mu}(j+1)$. This would imply that Property 3 is not satisfied for $\mu-1$, which contradicts the definition of μ .
- if $\mu = 1$ then agent b started $K_{1,2}(2(n+l)+1)$ before a has completed its j th fence $\Omega(j)$. This is a contradiction with Lemma 3.5

Hence agent b has completed $K_{\mu,\mu+1}(2(n+l)+1)$ while agent a was following $S_\mu(j+1)$. Similarly as before, agent b has also started following $K_{\mu,\mu+1}(2(n+l)+1)$ while agent a was following $S_\mu(j+1)$.

Hence agent b has followed the entire trajectory $K_{\mu,\mu+1}(2(n+l)+1)$ while a was following $S_\mu(j+1)$. However, by Definition 3.7, the trajectory $K_{\mu,\mu+1}(2(n+l)+1)$ contains $2(|B(4(2(n+l)+1))| + |A(8(2(n+l)+1))|)$ integral trajectories $X(2(n+l)+1)$. Moreover, according to Algorithm RV-asynch-poly, the number of nodes in trajectory $S_\mu(j+1)$ is equal to $2(|B(2(j+1))| + |A(4(j+1))|)$

which is at most $2(|B(2(2(n+l)+1))| + |A(4(2(n+l)+1))|)$ (recall that $j \leq 2(n+l)$ by Lemma 3.3). Thus, this would force a meeting because the number of integral trajectories $X(2(n+l)+1)$ in $K_{\mu,\mu+1}(2(n+l)+1)$ is larger than the number of nodes in $S_\mu(j+1)$, which is a contradiction.

Case 2. Property 2 is false for $i = \mu$. This implies that agent a completed $K_{\mu,\mu+1}(j+1)$ before agent b completed $S_\mu(2(n+l)+1)$. Hence agent a completed $K_{\mu,\mu+1}(j+1)$ while agent b was following $S_\mu(2(n+l)+1)$. Indeed, if agent a completed $K_{\mu,\mu+1}(j+1)$ before b started $S_\mu(2(n+l)+1)$, this would imply:

- if $\mu > 1$ then agent a started $K_{\mu,\mu+1}(j+1)$ before agent b completed $K_{\mu-1,\mu}(2(n+l)+1)$. Hence a had completed $S_\mu(j+1)$ before b completed $K_{\mu-1,\mu}(2(n+l)+1)$. This would imply that Property 4 is not satisfied for $\mu-1$, which contradicts the definition of μ .
- if $\mu = 1$ then agent a started $K_{1,2}(j+1)$ before b completed its $(2(n+l))$ th fence $\Omega(2(n+l))$ which contradicts Lemma 3.4.

Hence agent a completed $K_{\mu,\mu+1}(j+1)$ while b was following $S_\mu(2(n+l)+1)$. Similarly as before, agent a started $K_{\mu,\mu+1}(j+1)$ while b was following $S_\mu(2(n+l)+1)$.

Hence agent a has followed the entire trajectory $K_{\mu,\mu+1}(j+1)$ while b was following $S_\mu(2(n+l)+1)$. However, by Definition 3.7, the number of integral trajectories $X(j+1)$ (note that $X(j+1)$ is integral because $j \geq n+l+1$ in view of Lemma 3.3) in $K_{\mu,\mu+1}(j+1)$ is $2(|A(8(j+1))| + |B(4(j+1))|)$ which is at least $2(|A(8(n+l+2))| + |B(4(n+l+2))|)$ because $j \geq n+l+1$ in view of Lemma 3.3. Moreover, according to Algorithm RV-asynch-poly, the number of nodes in $S_\mu(2(n+l)+1)$ is less than $2(|A(4(2(n+l)+1))| + |B(2(2(n+l)+1))|) = 2(|A(8(n+l)+4)| + |B(4(n+l)+2))|)$. Thus, this would force a meeting because the number of integral trajectories $X(j+1)$ in $K_{\mu,\mu+1}(j+1)$ is larger than the number of nodes in $S_\mu(2(n+l)+1)$, which is a contradiction.

Case 3. Property 3 is false for $i = \mu$. This implies that agent b completed the first atom of $S_{\mu+1}(2(n+l)+1)$ before agent a completed $K_{\mu,\mu+1}(j+1)$. This implies that agent b completed the first atom of $S_{\mu+1}(2(n+l)+1)$ while a was following $K_{\mu,\mu+1}(j+1)$. Indeed, otherwise agent b would have completed $K_{\mu,\mu+1}(2(n+l)+1)$ before a completed $S_\mu(j+1)$ which would imply that Property 1 is false for μ . This is impossible by Case 1.

Hence agent b completed the first atom of $S_{\mu+1}(2(n+l)+1)$ while a was following $K_{\mu,\mu+1}(j+1)$. For the same reasons agent b also started the first atom of $S_{\mu+1}(2(n+l)+1)$ while a was following $K_{\mu,\mu+1}(j+1)$. Then while a was following $K_{\mu,\mu+1}(j+1)$, agent b either followed entirely the trajectory $A(8(n+l)+4)$ or followed entirely the trajectory $B(4(n+l)+2)$. Consequently, in view of Definitions 3.5 and 3.6, agent b must have followed trajectory $X(j+1, u)$ for every node u of the graph at least once (because $j \leq 2(n+l)$ by Lemma 3.3). Since $K_{\mu,\mu+1}(j+1)$ consists of repeating $X(j+1, v)$ for the same node v , agents would meet in view of Lemma 3.1, which is a contradiction.

Case 4. Property 4 is false for $i = \mu$. This implies that agent a completed the first atom of $S_{\mu+1}(j+1)$ before agent b completed $K_{\mu,\mu+1}(2(n+l)+1)$. This implies that agent a completed the first atom of $S_{\mu+1}(j+1)$ while b was following $K_{\mu,\mu+1}(2(n+l)+1)$. Indeed, otherwise agent a would have completed $K_{\mu,\mu+1}(j+1)$ before agent b completed $S_\mu(2(n+l)+1)$ which would imply that Property 2 is false for μ . This is impossible by Case 2.

Hence agent a completed the first atom of $S_{\mu+1}(j+1)$ while b was following $K_{\mu,\mu+1}(2(n+l)+1)$. For the same reasons agent a also started the first atom of $S_{\mu+1}(j+1)$ while b was following $K_{\mu,\mu+1}(2(n+l)+1)$. Then while b was following $K_{\mu,\mu+1}(2(n+l)+1)$, agent a either followed entirely the trajectory $A(4(j+1))$ or followed entirely the trajectory $B(2(j+1))$. Consequently, in view of Definitions 3.5 and 3.6, agent a must have followed trajectory $X(2(n+l)+1, u)$ for every

node u of the graph at least once (because $j \geq n + l + 1$ by Lemma 3.3). Since $K_{\mu, \mu+1}(2(n + l) + 1)$ consists of repeating $X(2(n + l) + 1, v)$ for the same node v , agents would meet in view of Lemma 3.1, which is a contradiction. \square

Theorem 3.1 *There exists a polynomial $\Pi(x, y)$, non decreasing in each variable, such that if two agents with labels L_1 and L_2 execute Algorithm RV-asynch-poly in a graph of size n , then their meeting is guaranteed by the time one of them performs $\Pi(n, \min(|L_1|, |L_2|))$ edge traversals.*

Proof: Let $m = \min(|L_1|, |L_2|)$. Let a be the agent with label L_1 and let b be the agent with label L_2 . Let M_a be the modified label of agent a and let M_b be the modified label of agent b . Let l be the length of the shorter of labels M_a, M_b . Hence $l = 2m + 2$. As observed before, the modified label of one agent cannot be a prefix of the modified label of the other. Hence there exists an integer $\lambda \geq 1$, such that the λ th bit of M_a is different from the λ th bit of M_b . Let t be the first time at which an agent finishes its $(2(n + l) + 1)$ th piece. By Lemma 3.6, if the agents have not met by time t , then one of them cannot have completed the first atom of $S_\lambda(k_1)$ as long as the other agent has not completed $K_{\lambda-1, \lambda}(k_2)$ (i.e. started $S_\lambda(k_2)$), for some $2(n + l) + 1 \geq k_1, k_2 \geq n + l + 2$. (Since $k_1, k_2 \geq n + l + 2$ and $\lambda \geq 1$, these objects must exist.)

First suppose that the λ th bit of M_a is 1. There are two possible cases.

- agent a follows the entire trajectory $B(2(j + 1))$ while agent b is following $S_\lambda(2(n + l) + 1) = A(8(n + l) + 4)^2$.

Since $j \geq n + l + 1$, by Definition 3.6 the trajectory $B(2(j + 1))$ contains $2|A(8j + 8)| \geq 2|A(8(n + l + 1) + 8)|$ integral trajectories $Y(2(j + 1))$. Moreover, according to Algorithm RV-asynch-poly, the number of nodes in $S_\lambda(2(n + l) + 1)$ is $2|A(8(n + l) + 4)|$. So, the trajectory $B(2(j + 1))$ contains more integral trajectories $Y(2(j + 1))$ than there are nodes in $S_\lambda(2(n + l) + 1)$, hence there is a meeting.

- agent b follows the entire trajectory $A(4(2(n + l) + 1))$ while agent a is following $S_\lambda(j + 1) = B(2(j + 1))^2$.

The trajectory $S_\lambda(j + 1)$ consists of repetitions of $Y(2(j + 1), v)$ for some node v . Since by Lemma 3.3, $j \leq 2(n + l)$, the trajectory $A(4(2(n + l) + 1))$, contains $Y(2(j + 1), u)$ for every node u of the graph, which implies a meeting by Lemma 3.1.

Next suppose that the λ th bit of M_a is 0. There are two possible cases.

- agent a follows the entire trajectory $A(4(j + 1))$ while agent b is following $S_\lambda(2(n + l) + 1) = B(2(2(n + l) + 1))^2$.

The trajectory $S_\lambda(2(n + l) + 1)$ consists of repetitions of $Y(4(n + l) + 2)$ for some node v . Since by Lemma 3.3, $j \geq n + l + 1$, the trajectory $A(4(j + 1))$, contains $Y(4(n + l) + 2, u)$ for every node u of the graph, which implies a meeting by Lemma 3.1.

- agent b follows the entire trajectory $B(2(2(n + l) + 1))$ while agent a is following $S_\lambda(j + 1) = A(4(j + 1))^2$.

By Definition 3.6 the trajectory $B(2(2(n + l) + 1))$ contains $2|A(16(n + l) + 8)|$ integral trajectories $Y(2(2(n + l) + 1))$. Moreover, since $j \leq 2(n + l)$, the number of nodes in $S_\lambda(j + 1)$

is $2|A(4(j+1))|$ i.e., at most $2|A(8(n+l)+4)|$. So, the number of integral trajectories $Y(2(2(n+l)+1))$ in $B(2(2(n+l)+1))$ is larger than the number of nodes in $S_\lambda(j+1)$, hence there is a meeting.

Hence in all cases agents meet by the time when the first of the agents completes its $(2(n+l)+1)$ th piece. Now the proof can be completed by the following estimates which are a consequence of the formulation of the algorithm and of the definitions of respective trajectories. (Recall that P is the polynomial describing the number of edge traversals in the trajectory obtained by Reingold's procedure.)

For any v , $|X(k, v)| \leq X_k^* = 2P(k) + 1$.

For any v , $|Q(k, v)| \leq Q_k^* = \sum_{i=1}^k X_i^*$.

For any v , $|Y(k, v)| \leq Y_k^* = 2P(k) \cdot Q_k^*$.

For any v , $|Z(k, v)| \leq Z_k^* = \sum_{i=1}^k Y_i^*$.

For any v , $|A(k, v)| \leq A_k^* = 2P(k) \cdot Z_k^*$.

For any v , $|B(k, v)| \leq B_k^* = 2A_{8k}^* \cdot Y_k^*$.

For any v , $|K(k, v)| \leq K_k^* = 2B_{8k}^* \cdot X_k^*$.

For any v , $|\Omega(k, v)| \leq \Omega_k^* = (2k-1)K_k^* \cdot X_k^*$.

For every integer $k > 0$, let T_k^* denote the number of nodes in a piece in iteration k of the repeat loop in Algorithm RV-asynch-poly. Let $N = 2(n+l)+1$. Recall that $l = 2m+2$. We have $T_k^* \leq N(2A_{4k}^* + 2B_{2k}^* + K_k^*)$. For any agent, the length of the trajectory it follows by the time it completes the $(2(n+l)+1)$ th piece is at most $\sum_{k=1}^N (T_k^* + \Omega_k^*)$. Let $\Pi(n, m) = \sum_{k=1}^N (T_k^* + \Omega_k^*)$. It follows from the above discussion that agents must meet by the time one of them performs $\Pi(n, m)$ edge traversals. Since T_k^* and Ω_k^* are polynomials in k , while N and l are polynomials in n and m , the function $\Pi(n, m)$ is a polynomial. Since the polynomial $P(k)$ is non-decreasing, Π is non-decreasing in each variable. This completes the proof. \square

4 Applications: solving problems for multiple asynchronous agents

In this section we apply our polynomial-cost rendezvous algorithm for asynchronous agents to solve four basic distributed problems involving multiple asynchronous agents in unknown networks. Agents solve these problems by exchanging information during their meetings. The scenario for all the problems is the following. There is a team of $k > 1$ agents having distinct integer labels, located at different nodes of an unknown network. The adversary wakes up some of the agents at possibly different times. A dormant agent is also woken up by an agent that visits its starting node, if such an agent exists. As before, each agent knows a priori only its own label. Agents do not know the size of the team and, as before, have no a priori knowledge concerning the network. The assumptions concerning the movements of agents remain unchanged. We only need to add a provision in the model specifying what happens when agents meet. (For rendezvous, this was the end of the process.) This addition is very simple. When (two or more) agents meet, they notice this fact and can exchange all previously acquired information. However, if the meeting is inside an edge, they continue the walk prescribed by the adversary until reaching the other end of the current edge. New knowledge acquired at the meeting can then influence the choice of the subsequent part of the routes constructed by each of the agents. It should be noted that the possibility of exchanging all current information at a meeting is formulated only for simplicity. In fact, during a meeting, our

algorithm prescribes the exchange of only at most k labels of other agents that the meeting agents have already heard of, their initial values in the case of the gossiping problem, and a constant number of control bits.

We now specify the four problems that we want to solve:

- **team size**: every agent has to output the total number k of agents;
- **leader election**: all agents have to output the label of a single agent, called the leader;
- **perfect renaming**: all agents have to adopt new different labels from the set $\{1, \dots, k\}$, where k is the number of agents;
- **gossiping**: each agent has initially a piece of information (value) and all agents have to output all the values; thus agents have to exchange all their initial information.

The cost of a solution of each of the above problems is the total number of edge traversals by all agents until they output the solution. Using our rendezvous algorithm we solve all these problems at cost polynomial in the size of the graph and in the smallest length of all labels of participating agents.

Let us first note that accomplishing all the above tasks is a consequence of solving the following problem: at some point each agent acquires the labels of all the agents *and is aware* of this fact. We call this more general problem Strong Global Learning (SGL), where the word “strong” emphasizes awareness of the agents that learning is accomplished.¹ Indeed, if each agent gets the labels of all the agents *and is aware* of it, then each agent can count all agents, thus solving **team size**, each agent can output the smallest label as that of the leader, thus solving **leader election**, each agent can adopt the new label i if its original label was i th in increasing order among all labels, thus solving **perfect renaming**, and each agent can output all initial values, thus solving **gossiping**, if we append in the algorithm for SGL the initial value to the label of each agent.

Hence it is enough to give an algorithm for the SGL problem, working at cost polynomial in the size of the graph and in the smallest length of all labels of participating agents. This is the aim of the present section. Notice that this automatically solves all distributed problems that depend only on acquiring by all agents the knowledge of all labels and *being aware of this fact*. (The above four problems are in this class.) We stress this latter requirement, because it is of crucial importance. Note, for example, that none of the above four problems can be solved even if agents eventually learn all labels but are never aware of the fact that no other agents are in the network. This detection requirement is non-trivial to achieve: recall that agents have no a priori bound on the size of the graph or on the size of the team.

We now describe Algorithm SGL solving the SGL problem at cost polynomial in the size of the graph and in the smallest length of all labels of participating agents. In this description we will use procedure RV-ASYNCH-POLY(L) to denote Algorithm RV-asynch-poly as executed by an agent with label L . For technical reasons, an agent with label L will execute RV-ASYNCH-POLY(L') for L' different from L , i.e., an agent will mimic the execution of Algorithm RV-asynch-poly by an agent with a label different from its own.

Algorithm SGL

¹Notice that the assumption that the number k of agents is larger than 1 is necessary. For a single agent neither SGL nor any of the above mentioned problems can be solved. Indeed, for example in an oriented ring of unknown size (ports 0,1 at all nodes in the clockwise direction), a single agent cannot realize that it is alone.

For ease of presentation we will define three states in which an agent can be. These states are *traveller*, *explorer* and *token*. Transitions between states depend on the history of the agent, and more specifically on comparing the labels exchanged during meetings.

The high-level idea of the algorithm is the following. An agent a with label L wakes up in state *traveller* and executes procedure RV-ASYNCH-POLY($L + 1$) until the first meeting when it meets either other agents in state *traveller* or in state *token*. Then, depending on the comparison of labels of the agents it meets, it transits either to state *token* or to state *explorer*. In the first case it terminates its current move and stays idle. In the second case it simulates procedure *EST* learning the map of the graph and in particular its size n . Then it executes procedure RV-ASYNCH-POLY(1) until it performed $\Pi(n, 1)$ edge traversals. Now it has met all agents in state *traveller*. At this point two consecutive DFS traversals of the already known graph permit it to learn all labels of participating agents and to convey this knowledge to all agents in state *token*. All other agents will in turn get this knowledge from these agents.

Below we specify what an agent a with label L does in each state and how it transits from state to state. Each agent has a set variable W , called its *bag*, initialized to $\{L\}$, where L is its label. At each point of the execution of the algorithm the value of the bag is the set of labels of agents that a has been informed about. More precisely, during any meeting of a with agents whose current values of their bags are W_1, W_2, \dots, W_i , respectively, agent a sets the value of its bag W to $W \cup W_1 \cup W_2 \cup \dots \cup W_i$. Notice that since each bag can be only incremented, the number of updates of each bag is at most $k - 1$, where k is the number of agents.

State *traveller*.

The agent a wakes up in this state and starts executing procedure RV-ASYNCH-POLY($L + 1$) until the first meeting. Suppose the first meeting is with a set Z of agents. If Z contains an agent in state *token*, then let b be the smallest agent in this state in set Z . Agent a transits to state *explorer* and considers b as its token. If Z does not contain any agent in state *token* but contains agents in state *traveller*, then let c be the smallest agent in this state in set Z . If a is smaller than c , then a transits to state *token* and all agents in state *traveller* from Z transit to state *explorer* considering a as their token. If a is larger than c , then a and all agents in state *traveller* from Z except c transit to state *explorer* and they consider c (which transits to state *token*) as their token. Finally, if Z consists only of agents in state *explorer*, then agent a ignores this meeting and continues executing procedure RV-ASYNCH-POLY($L + 1$) until the next meeting.

State *token*.

In this state the agent a completes the traversal of the current edge and remains idle at its extremity forever. As soon as it gets the information (from some agent in state *explorer*) that its current bag contains all labels of participating agents, agent a outputs the value of its bag.

State *explorer*.

When agent a transits to this state, it has just met an agent b in state *token* (or which has just transited to state *token*), that a considers as its token. The actions of agent a are divided into two phases.

Phase 1.

If the meeting was at a node, agent a performs the procedure *EST* with the token at this node. If the meeting was inside an edge e , agent a simulates procedure *EST* in a graph G' differing from the real graph by adding a node w of degree 2 inside edge e and treating the agent b as the token

residing at w . During the simulation, when a subsequently meets b , this can happen either inside edge e or in one of the extremities u, v of e . In the first case agent a behaves as if the meeting of the token were in w . In particular, if all meetings with b during the simulation of EST are inside e , agent a acts as follows after each meeting. It finishes the traversal of e going, say to u and then either simulates the next move by going to v if the simulated move is from w to v or by doing nothing, if the simulated move is from w to u . If some meeting with b during the simulation is at one of the extremities, a aborts the simulation and launches EST in the real graph with the token at this node. (In this case agent b playing the role of the token will stay idle forever at the node.) After completing Phase 1 agent a learns the complete map of the graph, and in particular its size n .

Phase 2.

Knowing the size n of the graph, agent a executes procedure RV-ASYNCH-POLY(1) until it performed $\Pi(n, 1)$ edge traversals. (Note that $|1| = 1$.) We will show that at this point it met all agents that were either still dormant or were in state *traveller*. The labels of all remaining agents are in the union of bags of all agents currently in state *token*. Agent a performs two consecutive DFS traversals of the graph (whose map is already known to it), traversing all of its edges. After the first DFS traversal agent a has in its bag the labels of all participating agents. During the second DFS traversal, all these labels are transmitted to all agents in state *token*, together with the information that this is the set of all labels. After completing the second traversal agent a outputs the value of its bag.

Theorem 4.1 *Upon completion of Algorithm SGL, each agent outputs the set of labels of all participating agents. The total cost of the algorithm is polynomial in the size of the graph and in the smallest length of all labels of participating agents.*

Proof: Let M be the smallest of all labels of participating agents. Consider an agent a with label L . Upon waking up the agent is in state *traveller* and starts executing procedure RV-ASYNCH-POLY($L + 1$). In view of Theorem 3.1, by the time it performs $\Pi(n, |M + 1|)$ edge traversals, it must meet some agent in state *traveller* or in state *token*. (Indeed, during this time interval, the agent with label M is either idle, or is executing procedure RV-ASYNCH-POLY($M + 1$), so if agent a has not met another agent in state *traveller* or in state *token* before, it must meet the agent with label M that is in one of these states.) At this meeting, agent a transits either to state *token* or to state *explorer*. In the first case agent a does not perform any further edge traversals. We will later show that at some point an agent in state *token* has all the labels in its bag and is aware of it, thus outputting the value of this bag. Now consider the second case, when agent a transited to state *explorer*. In this case agent a uses at most $T(EST(n + 1)) + T(EST(n))$ edge traversals in Phase 1: at most the cost of one aborted simulation of EST in a graph of size $n + 1$ and one execution of EST in a graph of size n . After completing Phase 1 agent a learns the complete map of the graph, and in particular its size n . In state *explorer* agent a starts Phase 2 by executing procedure RV-ASYNCH-POLY(1) until it performed $\Pi(n, 1)$ edge traversals in some time t'' . By this time agent a met all agents that are still in state *traveller*. This follows again from Theorem 3.1 (note that $L + 1 > 1$, for any label L). Hence at time t'' , agent a has in its bag the labels of all agents except possibly those that are already in state *token* or *explorer* at time t'' . Notice that the union of bags of all agents in state *token* at time t'' contains the labels of all agents in state *explorer* at time t'' . Now agent a performs twice a DFS traversal visiting all edges, which costs $O(n^2)$ edge traversals. By the end of the first DFS traversal, agent a must meet all agents that were in state

token at time t'' , as these agents never enter a different edge from the one where they transited to state *token*. Consequently, by the end of the first DFS traversal, the bag of agent a contains the labels of all agents, and a is aware of this fact. During the second DFS traversal, agent a transmits its bag to all agents currently in state *token* together with the information that this bag contains all labels. This permits all agents currently in state *token* to output the value of their bag which now contains all labels. Upon completion of the second DFS traversal agent a outputs the value of its bag which contains all labels.

Notice that after a given agent in state *explorer* has completed its second DFS traversal informing all agents currently in state *token* that their bag contains all labels, there can potentially still be some agents in state *traveller* that will later transit to state *token*. We have to argue that any such agent will at some point get this final information from some agent in state *explorer*. This is indeed the case: any agent D that transited to state *token* at some time τ must get this final information from some agent c that transited to state *explorer* at time τ (at the moment of meeting with D), because D is already in state *token* when c executes its second DFS traversal.

The number of edge traversals performed by any agent operating in a graph of size n can be bounded by $\Pi(n, |M + 1|) + \Pi(n, 1) + T(EST(n + 1)) + T(EST(n)) + O(n^2)$. This is a polynomial in n and $|M|$. Hence the total cost of Algorithm SGL executed in a graph of size n is polynomial in n and $|M|$. \square

5 Conclusion

We presented an algorithm for asynchronous rendezvous of agents in arbitrary finite connected graphs, working at cost polynomial in the size of the graph and in the length of the smaller label. In [18], where the exponential-cost solution was first proposed, the authors stated the following question:

Does there exist a deterministic asynchronous rendezvous algorithm, working for all connected finite unknown graphs, with complexity polynomial in the labels of the agents and in the size of the graph?

Our result gives a strong positive answer to this problem: our algorithm is polynomial in the *logarithm* of the smaller label and in the size of the graph.

In this paper we did not make any attempt at optimizing the cost of our rendezvous algorithm, the only concern was to keep it polynomial. Cost optimization seems to be a very challenging problem. Even finding the optimal cost of exploration of unknown graphs of known size is still open, and this is a much simpler problem, as it is equivalent to rendezvous of two agents one of which is inert.

We also applied our rendezvous algorithm to solve four fundamental distributed problems in the context of multiple asynchronous mobile agents. The cost of all solutions is polynomial in the size of the graph and in the length of the smallest of all labels.

References

- [1] H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg and R. Reischuk, Renaming in an asynchronous environment, *Journal of the ACM* 37 (1990), 524-548.

- [2] N. Agmon and D. Peleg, Fault-tolerant gathering algorithms for autonomous mobile robots, *SIAM J. Comput.* 36 (2006), 56-82.
- [3] S. Alpern, The rendezvous search problem, *SIAM J. on Control and Optimization* 33 (1995), 673-683.
- [4] S. Alpern, Rendezvous search on labelled networks, *Naval Research Logistics* 49 (2002), 256-274.
- [5] S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.
- [6] J. Alpern, V. Baston, and S. Essegaier, Rendezvous search on a graph, *Journal of Applied Probability* 36 (1999), 223-231.
- [7] E. Anderson and R. Weber, The rendezvous problem on discrete locations, *Journal of Applied Probability* 28 (1990), 839-851.
- [8] E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, *Proc. 14th Annual ACM Symp. on Computational Geometry* (1998), 365-373.
- [9] E. Anderson and S. Fekete, Two-dimensional rendezvous search, *Operations Research* 49 (2001), 107-118.
- [10] E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, A. Labourel, Almost optimal asynchronous rendezvous in infinite multidimensional grids, *Proc. 24th International Symposium on Distributed Computing (DISC 2010)*, *LNCS* 6343, 297-311.
- [11] V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, *Naval Research Logistics* 48 (2001), 722-731.
- [12] J. Chalopin, S. Das, A. Kosowski, Constructing a map of an anonymous graph: Applications of universal sequences, *Proc. 14th International Conference on Principles of Distributed Systems (OPODIS 2010)*, 119-134.
- [13] M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Solving the robots gathering problem, *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, 1181-1196.
- [14] R. Cohen and D. Peleg, Convergence properties of the gravitational algorithm in asynchronous robot systems, *SIAM J. Comput.* 34 (2005), 1516-1528.
- [15] R. Cohen and D. Peleg, Convergence of autonomous mobile robots with inaccurate sensors and movements, *SIAM J. Comput.* 38 (2008), 276-302.
- [16] A. Collins, J. Czyzowicz, L. Gasieniec, A. Labourel, Tell me where I am so I can meet you sooner. *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, 502-514.
- [17] J. Czyzowicz, A. Kosowski and A. Pelc, How to meet when you forget: Log-space rendezvous in arbitrary graphs, *Distributed Computing*, 25, 165-178 (2012)

- [18] J. Czyzowicz, A. Labourel, A. Pelc, How to meet asynchronously (almost) everywhere, *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, 22-30.
Journal version: ACM Transactions on Algorithms, to appear.
- [19] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vaccaro, Asynchronous deterministic rendezvous in graphs, *Theoretical Computer Science*, 355, 315-326 (2006)
- [20] A. Dessmark, P. Fraigniaud, D. Kowalski, A. Pelc. Deterministic rendezvous in graphs. *Algorithmica* 46 (2006), 69-96.
- [21] Y. Dieudonné, A. Pelc, Anonymous Meeting in Networks, *Proc. 24rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, 737-747.
- [22] Y. Dieudonné, A. Pelc, D. Peleg, Gathering despite mischief, *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, 527-540.
- [23] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous oblivious robots with limited visibility, *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, Springer LNCS 2010, 247-258.
- [24] P. Fraigniaud, E. Lazard, Methods and problems of communication in usual networks. *Discrete Applied Mathematics* 53 (1994), 79-133.
- [25] P. Fraigniaud, A. Pelc, Deterministic rendezvous in trees with little memory, *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, LNCS 5218, 242-256.
- [26] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, *Proc. 22nd Ann. ACM Symposium on Parallel Algorithms and Architectures (SPAA 2010)*, 224-232.
- [27] P. Fraigniaud, A. Pelc, Decidability classes for mobile agents computing, *Proc. 10th Latin American Theoretical Informatics Symposium (LATIN 2012)*, LNCS 7256, 362-374.
- [28] S. Guilbault, A. Pelc, Asynchronous rendezvous of anonymous agents in arbitrary graphs, *Proc. 15th International Conference on Principles of Distributed Systems (OPODIS 2011)*, LNCS 7109, 162-173.
- [29] D. Kowalski, A. Malinowski, How to meet in an anonymous network, in *13th Int. Colloquium on Structural Information and Comm. Complexity, (SIROCCO 2006)*, Springer LNCS 4056, 44-58.
- [30] E. Kranakis, D. Krizanc, and P. Morin, Randomized rendez-vous with limited memory, *Proc. 8th Latin American Theoretical Informatics (LATIN 2008)*, Springer LNCS 4957, 605-616.
- [31] E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, *Proc. 23rd Int. Conference on Distributed Computing Systems (ICDCS 2003)*, IEEE, 592-599.
- [32] W. Lim and S. Alpern, Minimax rendezvous on the line, *SIAM J. on Control and Optimization* 34 (1996), 1650-1665.

- [33] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, *Networks* 59 (2012), 331-347.
- [34] N.L. Lynch, *Distributed algorithms*, Morgan Kaufmann Publ. Inc., San Francisco, USA, 1996.
- [35] O. Reingold, Undirected connectivity in log-space, *Journal of the ACM* 55 (2008).
- [36] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, 599-608.
- [37] X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, *Proc. International Colloquium on Automata, Languages, and Programming (ICALP 1996)*, Springer LNCS 1099, 610-621.